Bash Shell Commands

A] General commands

B] File related commands

C] Process related commands

D] File transfer using UNIX

**A] General Commands**

**Ls**: lists your files   
**ls -l** :lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.   
**ls -a** : lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.   
There are many more options, for example to list files by size, by date, recursively etc.

**more filename** : shows the first part of a file, just as much as will fit on one screen. Just hit the space bar to see more or q to quit. You can use /pattern to search for a pattern.

**mkdir dirname** : make a new directory

**cd dirname** : change directory. You basically 'go' to another directory, and you will see the files in that directory when you do 'ls'. You always start out in your 'home directory', and you can get back there by typing 'cd' without arguments. 'cd ..' will get you one level up from your current position. You don't have to walk along step by step - you can make big leaps or avoid walking around by specifying [pathnames](http://mally.stanford.edu/~sr/computing/pathnames.html).

**pwd** : tells you where you currently are.

**B] File related commands**

1. **touch** : To create the empty file.

SYNTAX:

Touch [file name]

2. **cat** : Cat command is to create the file and allow the user to write the contents in it through standard input device. After writing the contents in the file press the ctrl + d keywords manually to save the contents and exit from the write state.

SYNTAX:

Cat [redirection operation][filename]

3. **cp** : This command is used to Copy files. If the last argument names an existing directory, cp copies each other given file into a file with the same name in that directory. Otherwise, if only two files are given, it copies the first onto the second. It is an error if the last argument is not a directory and more than two files are given. By default, it does not copy directories.

SYNTAX:

cp [options] source dest

4. **rm** : rm removes each specified file. By default, it does not remove directories. If a file is unwritable, the standard input is a tty, and the -f or –force option is not given, rm prompts the user for whether to remove the file. If the response does not begin with y or Y, the file is skipped.

SYNTAX:

rm [-dfirvR] [--directory] [--force] [--interactive] [--recursive]

[--help] [--version] [--verbose] name

5. **mv** : mv moves each other given file into a file with the same name in that directory. Otherwise, if only two files are given, it moves the first onto the second

SYNTAX:

mv [options] source dest

6. **ln** : Make links between files. There are two type of link are there hard link and soft link.

**Hard link**:

In case of hard link we can retrive our data after removal of all the link by lefting one. Although by removal of original file we can retrive our data also.

Inode no is same.

Permission are also same for all link.

No of link increased.

Hard link within the file is good.

**Soft link**:

We can’t retrieve our data back after removal of original file.

Inode no is different.

The permission are also different.

SYNTAX:

ln [options] source destination

ln [options/switches] source destination

7. **chmod** : Change the access permissions of files or directories. We can change the default permission by chmod. chmod changes the permissions of each given file according to mode, which can be either a symbolic representation of changes to make, or an octal number representing the bit pattern for the new permissions.

SYNTAX:

Chmod [permission change opration] [file or directory name]

8. **cmp** : Cmp command is used to compare two files

Syntax:

$cmp [filename1] [filename2]

9. **chown** : Change the ownership of one or more files to newowner OUTPUT. Newowner is either a user ID number or a login name located in /etc/passwd. The optional newgroup is either a group ID number (GID) or a group name located in the /etc/group file. When newgroup is supplied, the behavior is to change the ownership of one or more files to newowner and make it belong to newgroup.

SYNTAX:

chown [options] newowner[:newgroup] files

10. **chgrp** : Change the group of one or more files to newgroup. newgroup is either a group ID number or a group name located in /etc/group. You must own the file or be a privileged user to succeed with this command.

SYNTAX:

chgrp [options] newgroup files

11. **umask** : Sets a file-creation mask, umask sets the umask to mask & 0777.

RETURN VALUE

The previous value of the mask is returned.

SYNTAX:

umask [value to change the default value]

12. **file** : Classify the named files according to the type of data they contain. file checks the magic file (usually /etc/magic) to identify many common file types.

SYNTAX:

file [options] files

13. **comm** : Compare lines common to the sorted files file1 and file2.

SYNTAX:

comm [options] file1 file2

DESCRIPTION

Compare lines common to the sorted files file1 and file2. Three-column output is produced: lines unique to file1, lines unique to file2, and lines common to both files. comm is similar to diff in that both commands compare two files. In addition, comm can be used like uniq; that is, comm selects duplicate or unique lines between two sorted files, whereas uniq selects duplicate or unique lines within the same sorted file.

14. **wc** : Print the number of bytes, words, and lines in files. wc counts the number of bytes, whitespace-separated words, and newlines in each given file, or the standard input if none are given or when a file named – is given. It prints one line of counts for each file, and if the file was given as an argument, it prints the filename following the counts. If more than one filename is given, wc prints a final line containing the cumulative counts, with the filename total. The counts are printed in the order lines, words, bytes.

SYNTAX:

wc [-clw] [--bytes] [--chars] [--lines] [--words] [--help] [--version] [file...]

15. **tail** : tail prints the last part (10 lines by default) of each given file; it reads from standard input if no files are given or when a filename of – is encountered. If more than one file is given, it prints a header consisting of the file’s name enclosed in ==> and <== before the output for each file.

SYNTAX:

tail [-c [+]N[bkm]] [-n [+]N] [-fqv] [--bytes=[+]N[bkm]] [--lines=[+]N]

[--follow] [--quiet] [--silent] [--verbose] [--help] [--version] [file...]

tail [{-,+}Nbcfklmqv] [file...]

  eg. Tail -5 f1

16. **head** : Output the first part of files .head prints the first part (10 lines by default) of each given file; it reads from standard input if no files are given or when a filename of – is encountered. If more than one file is given, it prints a header consisting of the file’s name enclosed in ==> and <== before the output for each file.

SYNTAX:

head [–c N[bkm]] [–n N] [–qv] [--bytes=N[bkm]] [--lines=N] [--quiet] [--silent]

[--verbose] [--help] [--version] [file...]

head [–Nbcklmqv] [file...]

eg head -5 f1

17. **cut** : cuts selected fields of each line of a file.

List following –f is a list of fields assumed to be separated by a delimiter character.

Character following –d is delimiter.

Eg.

cut –f1,3 –d “:” prog1

output : This statement extracts the first and third field from file prog1. The delimiter between the fields is a “:”.

18. **grep**:

Print lines matching a pattern

SYNTAX:

grep [ –[[AB]]num ][–[CEFGVBchilnsvwx]][–e ] pattern j –ffile ][files... ]

DESCRIPTION

grep searches the named input files (or standard input if no files are named, or the filename – is given) for lines containing a match to the given pattern. By default, grep prints the matching lines. There are three major variants of grep, controlled by the following options:

–G Interpret pattern as a basic regular expression (see the list following this one). This is the default.

–E Interpret pattern as an extended regular expression.

–F Interpret pattern as a list of fixed strings, separated by newlines

**C] Process related commands**

When you execute a program on your UNIX system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.

Whenever you issue a command in UNIX, it creates, or starts, a new process. When you tried out the ls command to list directory contents, you started a process. A process, in simple terms, is an instance of a running program.

The operating system tracks processes through a five digit ID number known as the pid or process ID . Each process in the system has a unique pid.

Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any one time, no two processes with the same pid exist in the system because it is the pid that UNIX uses to track each process.

Starting a Process:

When you start a process (run a command), there are two ways you can run it:

Foreground Processes

Background Processes

**Foreground Processes:**

By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

The process runs in the foreground, the output is directed to screen, and if the ls command wants any input (which it does not), it waits for it from the keyboard.

While a program is running in foreground and taking much time, we cannot run any other commands (start any other processes) because prompt would not be available until program finishes its processing and comes out.

**Background Processes:**

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

The simplest way to start a background process is to add an ampersand ( &) at the end of the command.

$ls ch\*.doc &

This would also display all the files whose name start with ch and ends with .doc:

ch01-1.doc ch010.doc ch02.doc ch03-2.doc

ch04-1.doc ch040.doc ch05.doc ch06-2.doc

ch01-2.doc ch02-1.doc

Here if the ls command wants any input (which it does not), it goes into a stop state until I move it into the foreground and give it the data from the keyboard.

If you press the Enter key now, you see the following:

[1] + Done ls ch\*.doc &

$

The first line tells you that the ls command background process finishes successfully. The second is a prompt for another command.

**Listing Running Processes:**

It is easy to see your own processes by running the ps (process status) command as follows:

$ps

PID TTY TIME CMD

18358 ttyp3 00:00:00 sh

18361 ttyp3 00:01:31 abiword

18789 ttyp3 00:00:00 ps

One of the most commonly used flags for ps is the -f ( f for full) option, which provides more information as shown in the following example:

$ps -f

UID PID PPID C STIME TTY TIME CMD

amrood 6738 3662 0 10:23:03 pts/6 0:00 first\_one

amrood 6739 3662 0 10:22:54 pts/6 0:00 second\_one

amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh

amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f

Here is the description of all the fileds displayed by ps -f command:

|  |  |
| --- | --- |
| Column | Description |
| UID | User ID that this process belongs to (the person running it). |
| PID | Process ID. |
| PPID | Parent process ID (the ID of the process that started it). |
| C | CPU utilization of process. |
| STIME | Process start time. |
| TTY | Terminal type associated with the process |
| TIME | CPU time taken by the process. |
| CMD | The command that started this process. |

There are other options which can be used along with ps command:

|  |  |
| --- | --- |
| Option | Description |
| -a | Shows information about all users |
| -x | Shows information about processes without terminals. |
| -u | Shows additional information like -f option. |
| -e | Display extended information. |

**Stopping Processes**:

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when process is running in foreground mode.

If a process is running in background mode then first you would need to get its Job ID using ps command and after that you can use kill command to kill the process as follows:

$ps -f

UID PID PPID C STIME TTY TIME CMD

amrood 6738 3662 0 10:23:03 pts/6 0:00 first\_one

amrood 6739 3662 0 10:22:54 pts/6 0:00 second\_one

amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh

amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f

$kill 6738

Terminated

Here kill command would terminate first\_one process. If a process ignores a regular kill command, you can use kill -9 followed by the process ID as follows:

$kill -9 6738

Terminated

**Parent and Child Processes:**

Each unix process has two ID numbers assigned to it: Process ID (pid) and Parent process ID (ppid). Each user process in the system has a parent process.

Most of the commands that you run have the shell as their parent. Check ps -f example where this command listed both process ID and parent process ID.

**Zombie and Orphan Processes:**

Normally, when a child process is killed, the parent process is told via a SIGCHLD signal. Then the parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed before its child is killed. In this case, the "parent of all processes," init process, becomes the new PPID (parent process ID). Sometime these processes are called orphan process.

When a process is killed, a ps listing may still show the process with a Z state. This is a zombie, or defunct, process. The process is dead and not being used. These processes are different from orphan processes. They are the processes that has completed execution but still has an entry in the process table.

**Daemon Processes:**

Daemons are system-related background processes that often run with the permissions of root and services requests from other processes.

More clearly, a daemon is just a process that runs in the background, usually waiting for something to happen that it is capable of working with, like a printer daemon is waiting for print commands.

If you have a program which needs to do long processing then its worth to make it a daemon and run it in background.

**The top Command:**

The top command is a very useful tool for quickly showing processes sorted by various criteria.

It is an interactive diagnostic tool that updates frequently and shows information about physical and virtual memory, CPU usage, load averages, and your busy processes.

**Job ID versus Process ID:**

Background and suspended processes are usually manipulated via job number (job ID). This number is different from the process ID and is used because it is shorter.

In addition, a job can consist of multiple processes running in series or at the same time, in parallel, so using the job ID is easier than tracking the individual processes.

**D] File transfer using UNIX**

If you need to transfer files between your computer and a remote server foo.com

**FTP (file transfer protocol)**

This is mostly used to copy files between computers

ftp foo.com

You will have to enter your username and password when prompted.

If the file being transferred is a binary file (example PDF, PS, GIF, JPG, PNG files), type the following.

binary

To upload your file wolf.pdf to the directory in foo.com

cd /home/abc/backup

put wolf.pdf

To download the file abc.ps from the directory in foo.com

get abc.ps

If you have multiple files, use mget instead of get and mput instead of put.

To quit, type bye.

To check all the options using FTP, type man ftp from your shell prompt.

**SFTP (secure file transfer protocol)**

SFTP is an interactive file transfer program, similar to FTP, which performs all operations over an encrypted secsh transportTo connect to foo.com here is what you should do.

sftp foo.com

You will have to enter your username and password when prompted.

To upload your file wolf.pdf to the directory in foo.com

cd /home/ abc /backup

put wolf.pdf

To download the file abc.ps from the directory in foo.com

get abc.ps

If you want to download multiple files, say all PDF files,

get \*.pdf

To quit, type exit.

To check all the options using SFTP, type man sftp from your shell prompt.

**SCP (secure copy)**

SCP is used for single file transfers unlike SFTP or FTP, where once connected, you can carry out any number of transfers.

To upload the file wolf.pdf to the /home/abc/backup in the remote computer foo.com, Lets say the username and password for connecting to foo.com are us3r and p4ssword respectively, read ahead.

scp wolf.pdf us3r@foo.com:/home/abc/backup/

You will be prompted for your password, which you should enter. It uploads the file and quits automatically all in one operation.

To download the file abc.ps from the remote directory, here is what you must do.

scp us3r@foo.com:/home/abc/backup/abc.ps

If you want to upload the entire perl directory (recursively) here is what you do.

scp -r /home/wolf/perl us3r@foo.com:/home/abc/backup/

wget

To download the index.html page of gnome.org, here is what you type.

wget http://www.gnome.org/index.html